強化学習に基づいた分散チェックポイントの最適生成 (決定理論と最適化アルゴリズム)

AUTHOR(S):

岡村, 寛之; 西村, 祐樹; 土肥, 正

CITATION:

岡村, 寛之 ...[et al]. 強化学習に基づいた分散チェックポイントの最適生成 (決定理論と最適化アルゴリズム). 数理解析研究所講究録 2005, 1409: 10-20

ISSUE DATE:

2005-01

URL:

http://hdl.handle.net/2433/26153

数理解析研究所講究録 1409 巻 2005 年 10-20

強化学習に基づいた分散チェックポイントの最適生成

岡村寛之，西村祐樹，土肥正
Hiroyuki Okamura, Yuki Nishimura and Tadashi Dohi
Department of Information Engineering, Graduate School of Engineering,
Hiroshima University, Japan

1 はじめに

高度情報化が進んだ現代において，コンピュータは日常生活や社会活動などに大きく関わってきている．このようにコンピュータの使用に大きく依存する社会において銀行のオンラインバンキングシステム等のコンピュータシステムに障害が発生した場合，経済的損失や社会的混乱を招く恐れがある．そこで，このようなコンピュータシステムでは高い信頼性を保つことが要求されている．

システムの信頼性を向上させるためのアプローチは次の二つに大別できる．一つはシステム内にフォールトを作りこまないようにするフォールトアボイダンス技術である．この技術を用いることでシステムの信頼性は大きく向上するが，反面，大規模なシステムにおいてフォールトのない完全なシステムを作成することはきわめて難しいという欠点を持つ．

もう一つのアプローチとしては，システムに障害が発生することを前提とし，障害が発生した場合にもサービスを提供し続けるようにするフォールトトレラント技術である．代表的なものにチェックポイント，ソフトウェアレジュビネーションなどが挙げられ，本論文ではチェックポイントについて議論する．

チェックポイントとは以前に計算したデータを主記憶から安定な二次記憶媒体に保存する時期のことで，チェックポイントを配置することをチェックポインティングと呼ぶ．一般にコンピュータシステムに障害が発生すると処理を最初からやり直すのではなく，直前のチェックポイントまで後ろ向き回復（ロールバックリカバリ）を行い，そこから処理を再開することで障害の影響を軽減することができる．頻繁にチェックポイント生成を行うと，障害が発生した場合に元の状態まで戻る費用は小さくて済むが，データを記憶させるための費用は大きくなる．逆にチェックポイント生成を少なく行うと，障害が発生した場合に元の状態まで戻る費用が大きくなる．従って，それらのトレードオフを考えたチェックポイント間隔を求めることが必要となる

上記の問題に対して，Vaidya [1] は確率モデルを用いてチェックポイントモデルを表現し，計算ロスの最も少ない最適なチェックポイント間隔を解析的に導出している．しかしながら，この手法では予め障害発生時間間隔の確率分布を特定し，最適なチェックポイ

ント間隔を導出しなければならない．特に，確率分布を特定するためには，データが持つ統計的な特徴を考慮した上で経験的にいくつかの候補を選択し，その後に統計的検定作業を行う必要があるため，非効率的であるといわざるを得ない．また，Vaidya のモデルは単一プロセス処理を基礎としたものであるため，これを直接，分散処理を行うシステムに適用することができない．そこで本論文では，Vaidya のモデルを分散処理システムへ対応させるような修正を行い，さらに障害発生時間間隔に関する確率分布を特定することなく，データから直接最適なチェックポイント間隔を導出する手続きを提案する．

ここでは，強化学習と呼ばれる手法に基づいたチェックポインティングを提案する．強化学習とは，学習主体とシステムがおかれる環境との相互作用から最適な方策（ここではチェックポイント間隔）を学習するアルゴリズムである．強化学習の学習アルゴリズムとしてこれまでにいくつかの種類が提案されているが，本論文では Q 学習と呼ばれるアルゴリズムを用いる． Q 学習はマルコフ・セミマルコフ決定過程によって記述される環境において適用可能である．そこで本論文では，Vaidya [1] によって提案されたチェックポイントモデルをマルコフ決定過程によって再定式化し，Q 学習に基づいたチェックポインティングを提案する．

2 分散処理におけるチェックポイント

ここでは，プロセスに対するチェックポイントについて概説する．特に，分散処理に対するチェックポイント（分散チェックポイント）の問題点について言及する．

いま，複数のプロセスが協調して処理を行っている状況下で，あるプロセスが障害を起こした場合を考える．このとき，最も単純な対処としてすべてのプロセスが初めから実行をし直すことが考えられる．しかしながら，この方法は再実行に伴う無駄な処理（オーバーヘッド）が大きくかかり必ずしも効率的な対処法ではない．そこで，他の方法として，プロセスのある状態を記録しておき，障害が発生したならばその記録した時点から再度実行を行うことを考える．いま，適度な間隔でその記録がおこなわれるならば障害発生に対するオーバーヘッドを軽減することが可能となる．この記録するタイミングをチェックポイントと呼ぶ．

単一のプロセスあるいはシステムに対するチェックポイントを考える場合，チェックポイントを配置するタイミングは任意に行ってもよい．つまり，チェックポイントを配置することによるオーバーヘッドと障害によって行われるロールバックリカバリに要するオーバーヘッドを考慮して全体の処理時間を最小にするチェックポイント配置を行うことが可能である．しかしながら，分散処理におけるチェックポイントでは，複数のプロセスが協調して処理を行っているため

- ドミノ効果
- ライブロック

と呼ばれる問題が生じる．

ドミノ効果：図 1 はドミノ効果と呼ばれる現象が生じる例を示している．この例では p_1, p_2 という 2 つのプロセスが互いに通信（メッセージの送受信）を行いながら処理を行っている．図中ではあるプロセスから他のプロセスへの矢印がメッセージを表している．プロセス p_1 はチェックポイント cp_{11} を配置した後にメッセージ a を送信し，プロセス p_2 からメッセージ b を受信している．さらにプロセス p_1 はチェックポイント cp_{12} を配置し，メッセージ c をプロセス p_2 に送信している．いま，プロセス p_1 においてメッセージ c を送信後に障害が発生したとすると，プロセス cp_{12} へロールバックリカバリを行う．このとき，メッセージ c は送信しなかったことになるため，プロセス p_2 もメッセージ c を受信する以前の状態へ後退する必要ある．つまり，プロセス p_2 も最新のチェックポイント cp_{21} へロールバックする必要がある．しかしながら，このときメッセージ b もキャンセルされるためプロセス p_1 は更に前のチェックポイント cp_{12} まで後退する必要が生じる．このように，あるプロセスのロールバックリカバリによって次々にロールバックリカバリを繰り返す現象をドミノ効果と言う．

ライブロック：図 2 はライブロックの例を示している．ここの例ではプロセス p_1 に障害が発生しチェックポイント cp_1 にロールバックリカバリした後に再実行している．プロセス p_1 は，メッセージ a をプロセス p_2 に送信してから，メッセージ b の受信を行い，その後の処理を行っている．ここで，プロセス p_1 がロールバックリカバリすることによって，メッセージ a がキャンセルされるためにプロセス p_2 はチェックポイント cp_2 へ後退する．しかしながら，このときメッセージ b がキャンセルされるため，プロセス p_1 は再度 cp_1 にロールバックし，再実行される．また，プロセス p_2 も同様に，cp_2 へ後退し再実行する．このように，ロールバックリカバリと再実行が繰り返される現象をライブロックと言う．

これらの現象を防ぐために，分散処理においては (i) チェックポイント取得と (ii) プロセスの再実行の開始について同期をとる必要がある．このため，同期をとるべきプロセス数に依存したオーバーヘッドがかかることが考えられる．

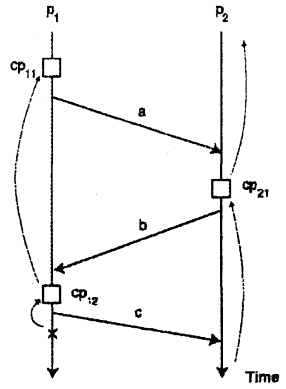

Figure 1: ドミノ効果の例.

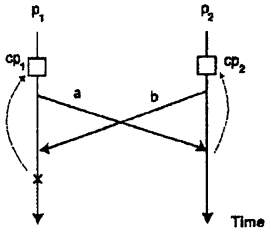

Figure 2: ライブロックの例.

3 マルコフ決定過程によるチェックポイントモデル

3.1 モデルの記述

Vaidya [1] は，単一プロセスにおけるチェックポインティング制御について以下のモデルを用いて考察を行っている．単一プロセスの処理時間はあるプロセス終了確率に従って終了し，システム障害は同次ポアソン過程に従って発生する．障害が発生すると直前のチェックポイントまでロールバックリカバリを行い処理の再実行がなされる．このとき元の状態まで回復するのに必要な時間をリカバリオーバーヘッドと呼び，直前のチェックポイントから障害が発生するまでの計算量が多いほどその時間は大きくなると仮定する．

本稿では Vaidya [1] のチェックポイントモデルを分散処理におけるチェックポイントを表現するように修正する．分散処理は互いに自律的に処理を行うことが前提であるため，単一プロセスに対するふるまいに着目する．また，他のプロセスに関する情報はわからないものとする．プロセスは正常な状態から稼動を開始し，メッセージの送信あるいは受信の直後にチェックポイントを生成するかどうかの決定を行う．この時点を決定点と呼ぶ．あるプロセスがチェックポイントを生成する場合，これまでに受信したプロセスとの同期をとる必要がある．そのため受信したプロセス数 m に依存した関数 $\tau(m)$ で表

されるオーバーヘッドがかかるものとする．また，チェックポイントを生成しない場合プロセスはデータ処理を継続する．単一のプロセスに対する障害はパラメータ λ (>0) のポアソン過程に従って発生する．分散処理では，これまでにメッセージを受信したプロセスに障害が発生した場合もロールバックリカバリを行う必要があるため，単一のプロセスに対する実質の障害発生率は $(m+1)\lambda$ である．さらに，ロールバックリカバリ後の再実行に関してもチェックポイント時と同様に送信したプロセスと同期をとる必要があるため，送信したプロセス数 n に依存したオーバーヘッド $R(n)$ がかかる．また，再実行によって障害が発生する直前までの回復には時間 $Z(n,m)$ を要し，この間チェックポイントの生成を行わないものと仮定する．また，回復完了時点が次の決定点となる．単一プロセスの処理はパラメータ α の指数分布に従う．さらに，他のプロセスへの送信および他のプロセスからの受信はパラメータ γ および θ のポアソン過程に従って発生する（図 3 参照）．

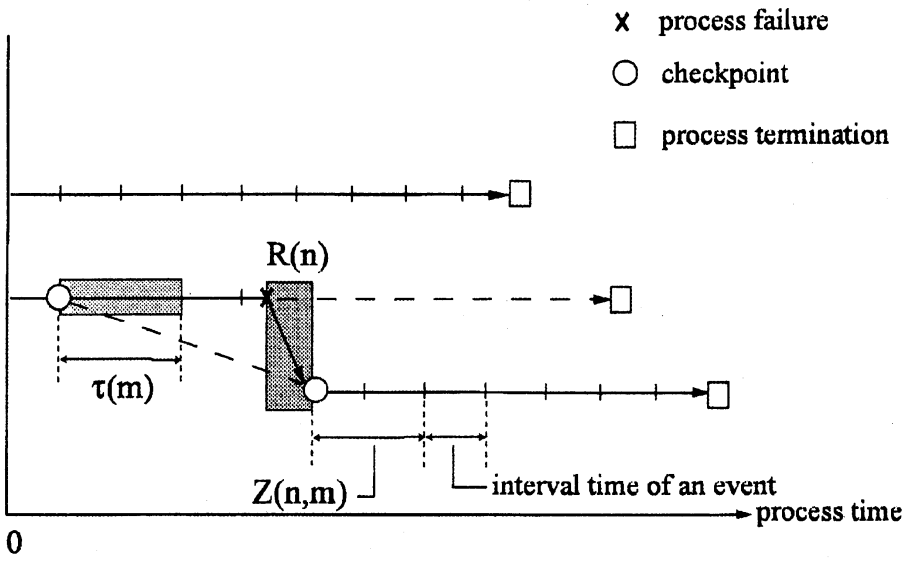

Figure 3: チェックポイント生成の挙動例.

また，上記のモデルにおいて以下の 5 つの状態を定義しその状態遷移を図 4 に示す．

State S0 ：正常稼動状態

State S1 ：チェックポイント生成状態

State S2 ：ロールバックリカバリ状態

State S3 ：処理再実行状態

State S4 ：プロセスの終了状態

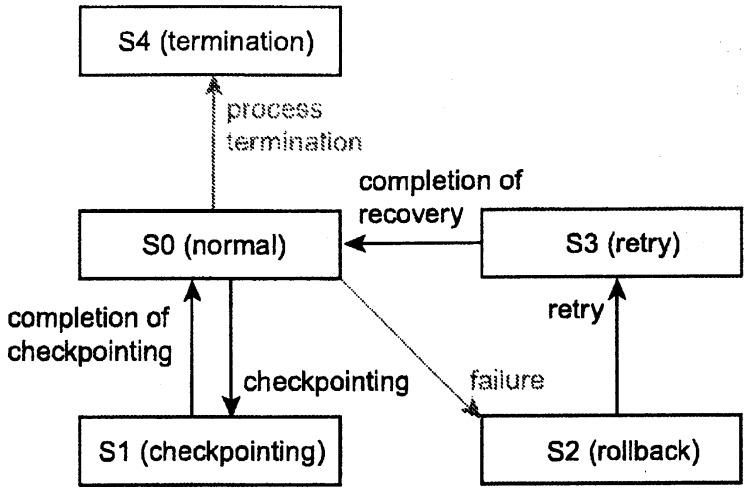

Figure 4: 状態遷移図.

3.2 マルコフ決定過程による定式化

マルコフ決定過程における状態集合および決定集合を以下のように表現する.

状態集合： $S := \{0, 1, 2, \cdots\} \times \{0, 1, 2, \cdots\}$

決定集合： $A(s) = \{\mathrm{cnt}, \mathrm{chk}\}$

ここでは，直前のチェックポイントから送信したメッセージ数と受信したメッセージ数をマルコフ決定過程における状態と定義し，その集合を状態集合としている．それぞれの状態において選択可能な決定は cnt と chk であり，それぞれ「チェックポイントを生成しない」と「チェックポイントを生成する」の決定に対応する．チェックポイント生成に関する評価尺度として，本論文では単一プロセスにおける総期待処理時間を考える．つまり，リカバリオーバーヘッドやチェックポイントオーバーヘッドも含めた総時間を考え，これを最小にする最適なチェックポイント配置方策を考える.

最適方策を導くために以下のような諸量を定義する.

$Q_n(n, m)$ ：直前のチェックポイントから n 個のメッセージを送信および m 個のメッセージを受信した状態でチェックポイントを生成しない決定を行い，それ以後最適な方策をとり続けた場合の総期待処理時間

$Q_c(n, m)$ ：直前のチェックポイントから n 個のメッセージを送信および m 個のメッセージを受信した状態でチェックポイントを生成する決定を行い，それ以後最適な方策

をとり続けた場合の総期待処理時間

$V(n,m)$ ：直前のチェックポイントから n 個のメッセージを送信および m 個のメッセージを受信した状態において最適な方策を選択した場合の最小総期待処理時間

このとき，以下の最適性方程式を得る．

$$V(n,m) = \min\{Q_n(n,m), Q_c(n,m)\}, \tag{1}$$

$$\begin{aligned} Q_n(n,m) = {} & \frac{1}{\nu_{n,m}} + \frac{(m+1)\lambda}{\nu_{n,m}}(R(n)+Z(n,m)+V(n,m)) \\ & + \frac{\gamma}{\nu_{n,m}}V(n+1,m) + \frac{\theta}{\nu_{n,m}}V(n,m+1) \\ & + \frac{nc}{\nu_{n,m}}(\tau(m)+V(0,0)) \end{aligned} \tag{2}$$

$$Q_c(n,m) = \tau(m) + Q_n(0,0). \tag{3}$$

ここで，$\nu_{n,m} = (m+1)\lambda + \alpha + \gamma + \theta + nc$ であり，c は他のプロセスのチェックポイントによって発生するチェックポイント処理の発生率である．これは本来，他のプロセスのチェックポイント方策に依存する値であるがここでは簡単のため，送信したプロセス数 n に依存する値として表現している．また，$Z(n,m)$, $R(n)$ および $\tau(m)$ は

$$Z(n,m) = c_z(e^{\eta_z nm} - 1) \tag{4}$$

$$R(n) = \xi_r + c_r(1 - e^{-\eta_r n}), \tag{5}$$

$$\tau(m) = \xi_\tau + c_p(1 - e^{-\eta_\tau m}) \tag{6}$$

とする．

4 強化学習によるチェックポイント生成

強化学習 [2] とは，エージェント（学習と意思決定を行うもの）が試行錯誤的に環境（制御する対象）との相互作用を学習して，環境に適応する（最適な制御を行う）ための方法論である．ニューラルネットワークのような教師付き学習と異なり，明示的な行動選択を示す教師が存在しない．その代わりにエージェントは環境から報酬（費用）という情報を得ることができ，その情報を基にして環境を支配するパラメータを学習する．強化学習を適用できる環境は，一般に次の性質を持つ．(1) エージェントは環境に関する知識を予め持たない，(2) 環境の状態遷移と報酬（費用）の発生は確率的である，(3) 報酬（費用）は状態遷移を繰り返すことで発生する．これらの性質は，環境の動的な特性をMDPによってモデル化可能であることを示唆しており，強化学習はMDPによるモデル化と密接な関係がある．

ここではチェックポイント生成アルゴリズムに対して強化学習を適用する．具体的な強化学習による学習アルゴリズムとして，これまでにさまざまなものが提案されている．本稿では特に，代表的なQ学習と呼ばれる強化学習アルゴリズムを適用する．Q学習は強化学習の代表的な手法であり，MDPによる定式化と深く関連している．MDPによる定式化では，ある状態sで行動aを選択し，以降の選択では最適な行動を選択し続けたときの値$Q(\cdot,\cdot)$を基に最適性方程式を考える．この値はQ値と呼ばれる．Q学習は，試行錯誤を繰り返しながらQ値の推定を行うアルゴリズムである．

具体的に，チェックポイント生成モデルに対するQ学習のアルゴリズムは以下の通りである．

Step 1：現在時刻$t(>0)$においてシステムの状態s_t（何番目の決定点であるか）を観測する．

Step 2：任意の行動選択法に従って行動a_t（処理の継続あるいはチェックポイント生成）を実行する．

Step 3：実行から$u(>0)$時間経過後，再びシステムの状態s_{t+u}（何番目の決定点であるか）を観測する．このとき実際の無駄時間の発生記録$X(u)$を記録する．

Step 4：以下の更新式によりQ値を更新する．

$$\begin{aligned} Q(s_t,a_t) \longleftarrow\ & (1-\beta)Q(s_t,a_t) \\ & +\beta\Big\{X(u)+\min_{\hat{a}} Q(s_{t+u},\hat{a})\Big\} \end{aligned} \tag{7}$$

ここで，$0<\beta\leq 1$は学習率と呼ばれる．

Step 5：現在時刻を$t+u$としてStep1へ戻る．

上記のアルゴリズムにおけるStep2では，Q値に基づいて行動を選択する方法を定める必要がある．本論文では確率的な行動選択法としてϵ-greedy選択を用いる[2]．

ϵ-greedy選択：$0<\epsilon<1$の確率でランダムに行動を選択する．それ以外では現在状態sに対応したQ値が最小となる行動を選択する．

5 数値例

ここでは，シミュレーションを通じてMDPによる厳密解とQ学習による最適制御の比較を行う．数値例におけるパラメータは以下の通りである．

Table 1: 最適方策（0:チェックポイントしない，1:チェックポイントする）.

		n										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	1	1	1	1	1	1	1	1	1
	1	0	0	0	0	0	0	0	1	1	1	1
	2	0	0	0	0	0	0	1	1	1	1	1
	3	0	0	0	0	0	1	1	1	1	1	1
m	4	0	0	0	0	1	1	1	1	1	1	1
	5	0	0	0	1	1	1	1	1	1	1	1
	6	0	0	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1

Table 2: 最小総期待処理時間.

		n										
		0	1	2	3	4	5	6	7	8	9	10
	0	39.79	40.68	40.79	40.79	40.79	40.79	40.79	40.79	40.79	40.79	40.79
	1	41.88	43.63	44.72	45.49	46.09	46.60	47.01	47.11	47.11	47.11	47.11
	2	43.30	45.18	46.45	47.42	48.24	48.97	49.44	49.44	49.44	49.44	49.44
	3	44.68	46.52	47.82	48.89	49.78	50.30	50.30	50.30	50.30	50.30	50.30
m	4	46.21	47.93	49.20	50.19	50.61	50.61	50.61	50.61	50.61	50.61	50.61
	5	47.89	49.44	50.48	50.73	50.73	50.73	50.73	50.73	50.73	50.73	50.73
	6	49.56	50.74	50.77	50.77	50.77	50.77	50.77	50.77	50.77	50.77	50.77
	7	50.78	50.78	50.78	50.78	50.78	50.78	50.78	50.78	50.78	50.78	50.78
	8	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79
	9	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79
	10	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79	50.79

$$\lambda = 0.001, \quad \alpha = 0.1 \quad \gamma = 1.0$$
$$\theta = 1.0 \quad c = 0.2 \quad c_r = 10$$
$$c_p = 10 \quad c_z = 0.5 \quad \eta_r = 1.0$$
$$\xi_r = 1.0 \quad \eta_\tau = 1.0 \quad \xi_\tau = 1.0$$
$$\eta_z = 1.0$$

式 (1)–(3) を Value Iteration Method によって解くことで，各状態における最適方策およびその時の総期待終了時間に関する結果が得られる．表 1 と表 2 はそれぞれ n 個のメッセージを送信し，m 個のメッセージを受信している場合の最適方策をおよび最小総期待処理時間を示している．

これらの表から $m = 0$ の場合を除いて最適なチェックポイント方策は $n + m$ の値がある値以下ならばチェックポイントを行わない，ある値を超える場合はチェックポイント

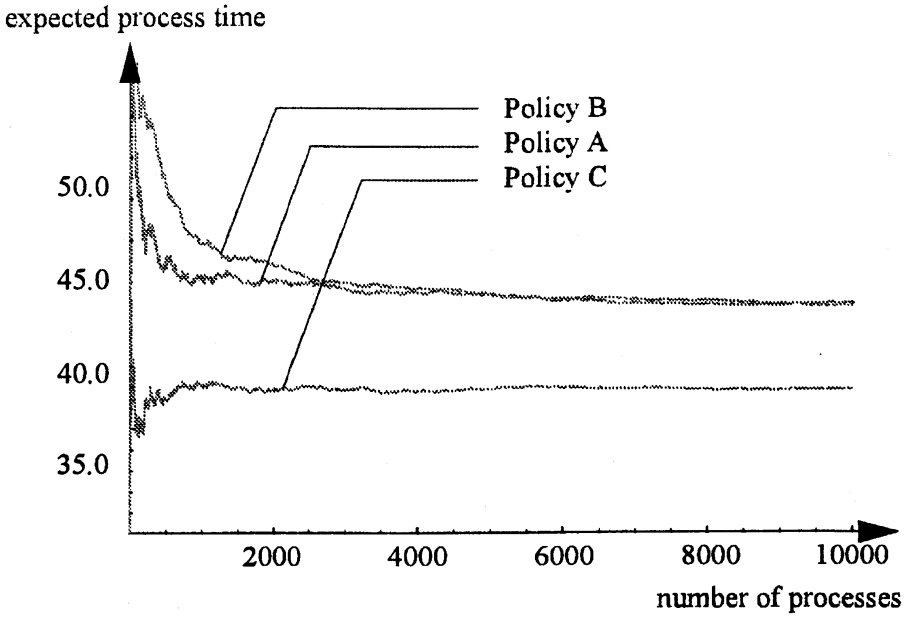

Figure 5: 方策の比較.

を行うことが最適となるような構造を持つことがわかる．しかしながら，現実的な状況において，λ などのパラメータを推定することは非常に困難であるため，前節で述べた強化学習を適用することを試みる．

強化学習によるチェックポインティングの有効性を検証するため，以下の 3 つの方策を適応した．

方策 A：Q 学習による制御を行う ($\epsilon = 0.0$).

方策 B：Q 学習による制御を行う ($\epsilon = 0.01$).

方策 C：MDP に基づいた最適制御を行う．

図 5 は上記の 3 つの方策を適用した際の結果である．ここでは横軸にシミュレーションを行ったプロセス数，縦軸はそれぞれのプロセス処理時間の合計をプロセス数で割った算術平均（総期待処理時間の推定値）を示している．この図から，MDP から得られた解析的に最適な方策は 1000 プロセス処理を行った時点でほぼ理論的に最小な総期待処理時間に収束していることがわかる．一方，強化学習による制御は全体的に理論値よりも長い時間かかる傾向にある．また，収束時間に関して 10000 プロセス終了した時点でも緩やかに理論値へ近づいていく傾向が見られ，収束が遅いことが見て取れる．しかしながら，全体としては最適な制御を行った場合の処理時間と比較して，相対誤差で 12% 程

度でありノンパラメトリックに最適解を求める手法としては比較的良好な結果が得られている.

6 今後の課題

本稿では分散処理におけるチェックポイント方策を導出するための強化学習アルゴリズムについて考察した. また, マルコフ決定過程における解析的に最適な方策と比較することによって強化学習の有効性を検証した. 今後は, ベイズ学習によるパラメータ推定および最適方策の導出アルゴリズムについて検討する. また, 実際の分散処理環境をシミュレートし, 提案したチェックポイントアルゴリムが効果的に機能することを検証する. また, ソフトウェアレジュビネーション方策 [3] をチェックポイントを同時に考慮した自律的なアルゴリズムについても検討する予定である.

References

[1] N. H. Vaidya, Impact of checkpoint latency on overhead ratio of a checkpointing scheme, *IEEE Transactions on Computers*, vol. 46, no. 8, pp. 942–947 (1997).

[2] R. S. Sutton and A. G. Barto（三上貞芳, 皆川雅章（共訳））: 「強化学習」, 森北出版 (2002).

[3] V. Castelli and others, Proactive management of software aging, *IBM J.Res.& Dev.*, vol. 45, no. 2, pp. 311–332 (2001).